**from** typing **import** List  
**import** typing  
**import** random  
**from** copy **import** deepcopy  
**import** base64  
  
**def** **generateRandomBinaryString**(p):  
 key1 = ""  
 **for** i **in** range(p):  
 temp = str(random.randint(0, 1))  
 key1 += temp  
 **return**(key1)  
  
  
**class** **Node**:  
 **def** **\_\_init\_\_**(self, left, right, value: str)-> **None**:  
 self.left: Node = left  
 self.right: Node = right  
 self.value = value  
  
 **def** **calculateHash**(s):  
 s = str(s)  
 s = s.encode("utf-8")  
 s = base64.b64encode(s)  
 **return** s  
  
 **def** **decodeHash**(s):  
 s = base64.b64decode(s)  
 s = s.decode("utf-8")  
 **return** s  
  
**class** **MerkleTree**:  
 **def** **\_\_init\_\_**(self, values: List[str])-> **None**:  
 self.\_\_buildTree(values)  
  
 **def** **\_\_buildTree**(self, values: List[str])-> **None**:  
 leaves: List[Node] = [Node(None, None, Node.calculateHash(e)) **for** e **in** values]  
 **if** len(leaves) % 2 == 1:  
 leaves.append(leaves[-1:][0])  
 self.root: Node = self.\_\_buildTreeRec(leaves)  
  
 **def** **\_\_buildTreeRec**(self, nodes: List[Node])-> Node:  
 half: int = len(nodes) // 2  
  
 **if** len(nodes) == 2:  
 **return** Node(nodes[0], nodes[1], Node.calculateHash(nodes[0].value + nodes[1].value))  
  
 left: Node = self.\_\_buildTreeRec(nodes[:half])  
 right: Node = self.\_\_buildTreeRec(nodes[half:])  
 value: str = Node.calculateHash(left.value + right.value)  
 **return** Node(left, right, value)  
  
 **def** **printTree**(self)-> **None**:  
 self.\_\_printTreeRec(self.root)  
  
 **def** **findDirtChunk**(self, merkelNode, merkel2Node)-> **None**:  
 **if** merkelNode != None:  
 **if**(merkelNode.value != merkel2Node.value):  
 print("Original Chunk: " + str(Node.decodeHash(merkelNode.value)))  
 print("Modified Chunk: " + str(Node.decodeHash(merkel2Node.value)))  
 self.findDirtChunk(merkelNode.left, merkel2Node.left)  
 self.findDirtChunk(merkelNode.right, merkel2Node.right)  
   
  
 **def** **\_\_printTreeRec**(self, node)-> **None**:  
 **if** node != None:  
 print(node.value)  
 self.\_\_printTreeRec(node.left)  
 self.\_\_printTreeRec(node.right)  
  
 **def** **getRootHash**(self)-> str:  
 **return** Node.decodeHash(self.root.value)  
   
elems = generateRandomBinaryString(8)  
elems\_list = list(elems)  
mtree = MerkleTree(elems\_list)  
  
modified\_elems = deepcopy(elems)  
modifiedElems\_list = list(modified\_elems)  
**if**(modified\_elems[len(elems)//2] == 1):  
 modifiedElems\_list[len(elems)//2] = '0'  
**else**:  
 modifiedElems\_list[len(elems)//2] = '1'  
modified\_elems = "".join(modifiedElems\_list)  
mtree\_2 = MerkleTree(modifiedElems\_list)  
  
hash1 = str(mtree.getRootHash())  
hash2 = str(mtree\_2.getRootHash())  
  
hash1\_list = set(hash1)  
hash2\_list = set(hash2)  
  
print(f"Original String: {str(elems\_list)}")  
print(f"Modified String: {str(modifiedElems\_list)}")  
  
print(f"Hash for original string: {hash1}\n")  
print(f"Hash for modified string: {hash2}\n")  
**if**(hash1 == hash2):  
 print("Yes the hashes match!")  
**else**:  
 print("No the hashes don't match!\n")  
 print("Now discovering the faulted chunk in the modified merkel tree....\n")  
 mtree\_2.findDirtChunk(mtree.root, mtree\_2.root)

